Continuously Delivering All the Things

Container Solutions

info@container-solutions.com
www.container-solutions.com

Who Am I?

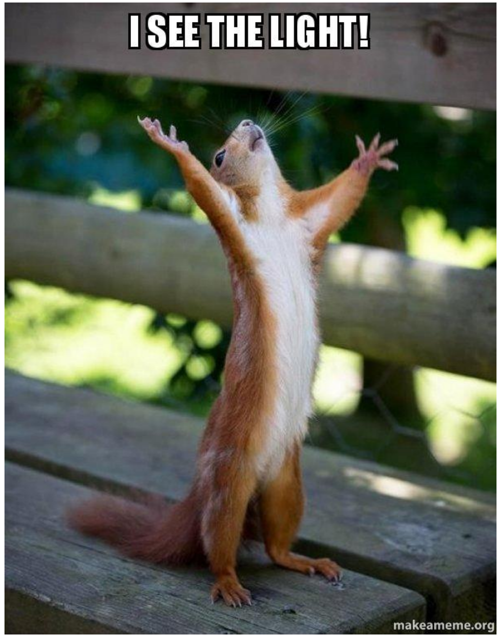
- Consultant for Container Solutions
- Main focus of moving companies over to the Cloud/Kubernetes and containers

The Problem

- Infrastructure comes with a variety of responsibilities:
  - Audit Trail
  - Stability
  - Security
  - DR Strategy
  - etc...

GitOps

- Version controlled state of your infrastructure
- Each commit can be signed by someone
- Changes only happen via pipeline
- Declarative not imperative
- Collaboration on pull requests



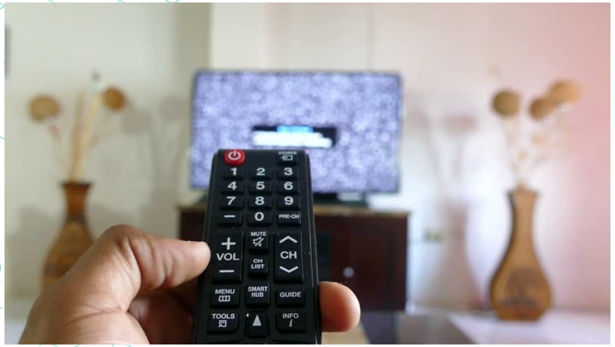
Imperative Vs Declarative

Imperative:

Step One: …..

Step Two: …..

Step Three: ….



VS

Declarative:

This is what I want.

Now make it happen

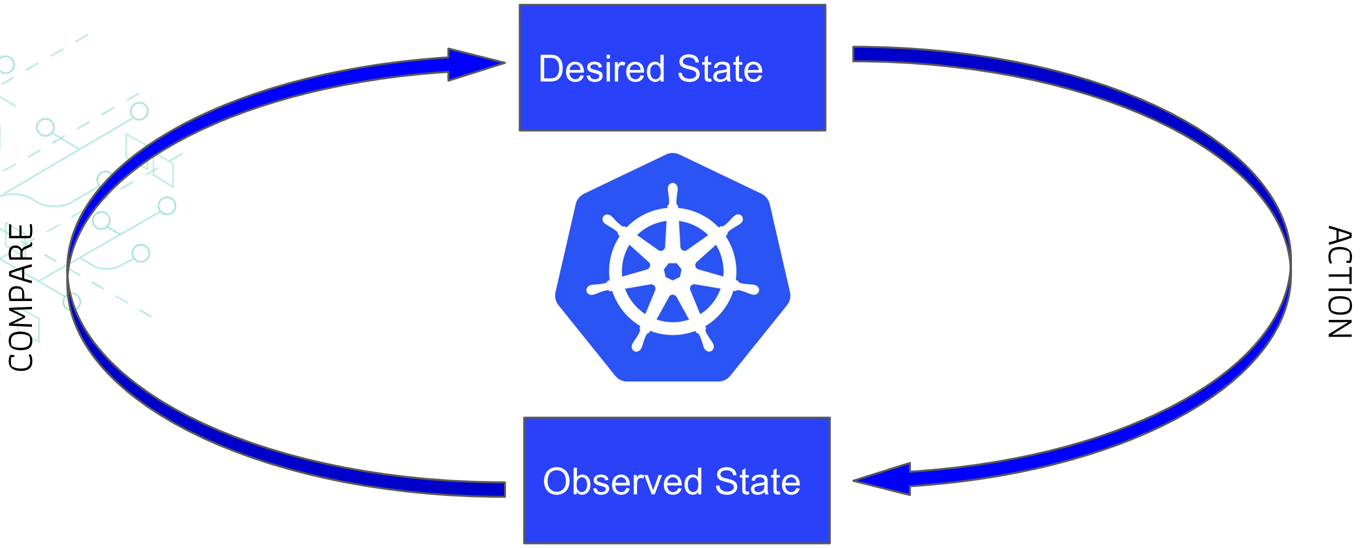
Collaboration on Infrastructure

A fresh pair of eyes has solved many tired problems

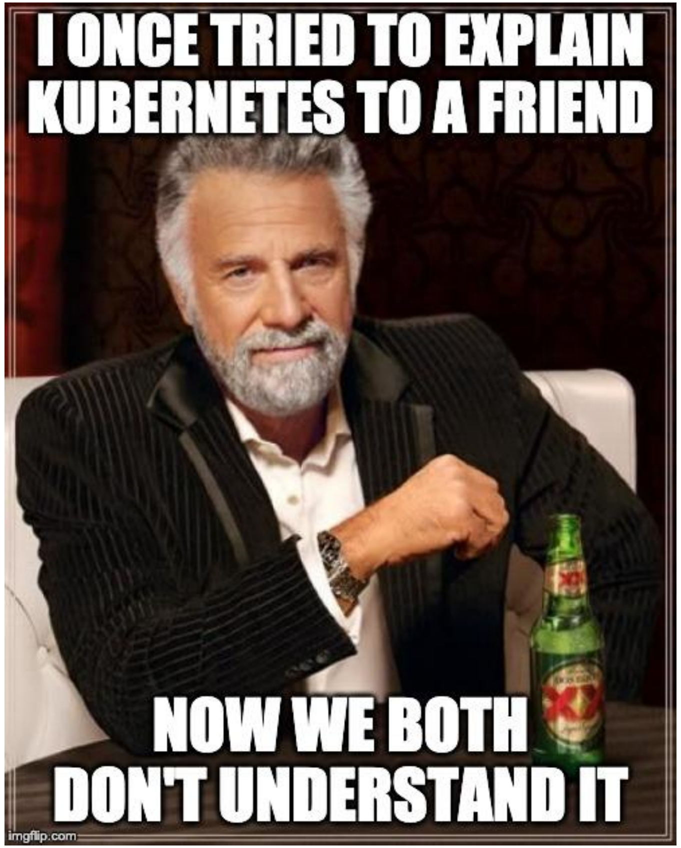
What About Security?

- Your pipeline and a trusted few actually have access to make changes
- Permissions are based on your Git access
- Every change can be tracked to the person who made it (via the commit) and you can see who approved it (via the pull request)

Reconciliation Loop



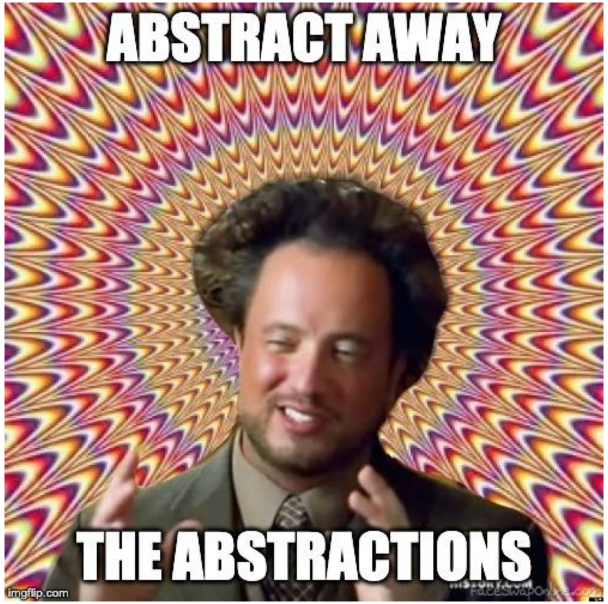
Kubernetes Can Get Complex


I ONCE TRIED TO EXPLAIN KUBERNETES TO A FRIEND
NOW WE BOTH DON'T UNDERSTAND IT
imgflip.com

Complexity has to go somewhere.....

But not always in your pipeline

- Put your complexity where there is a lot of flexibility
- Be careful to not have to many abstracted layers!!!!!



REPO == Real Life

GitLab =

Treat your Infrastructure Like an App

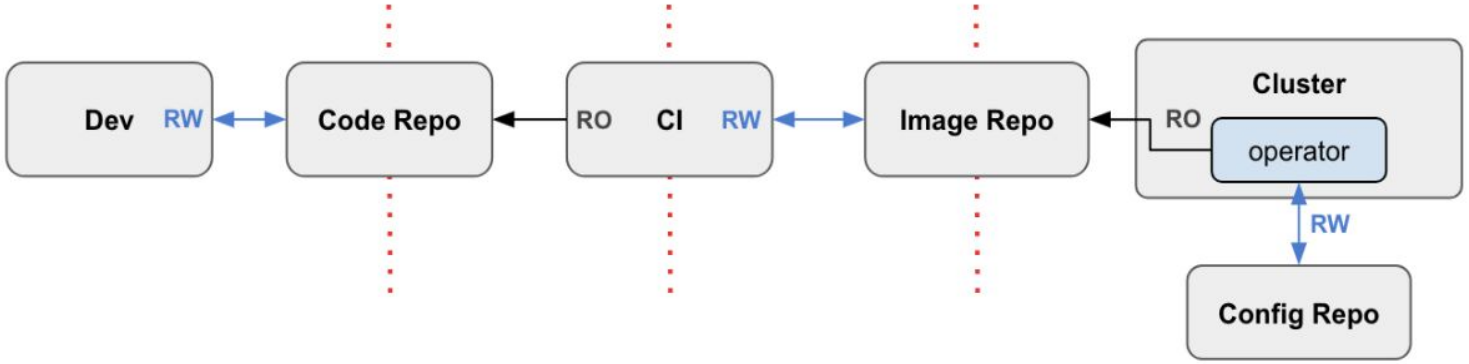
- Run tests!!
- Review changes
- Stateless
- Health checks
- The list goes on

Push VS Pull

PUSH
IF THAT DOESN'T WORK
PULL
IF THAT DOESN'T WORK EITHER
THAT'S BECAUSE WE'RE CLOSED

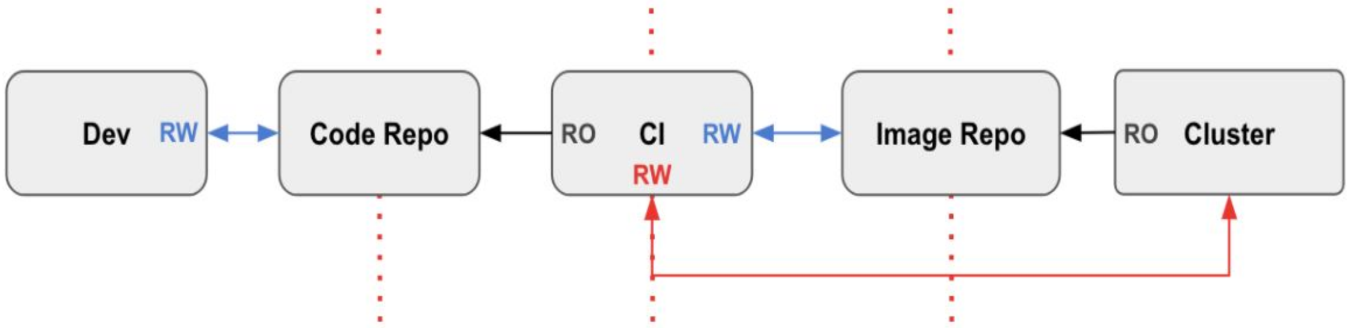
Pull Model

- Pull is generally the preferred methodology as everything is in your cluster



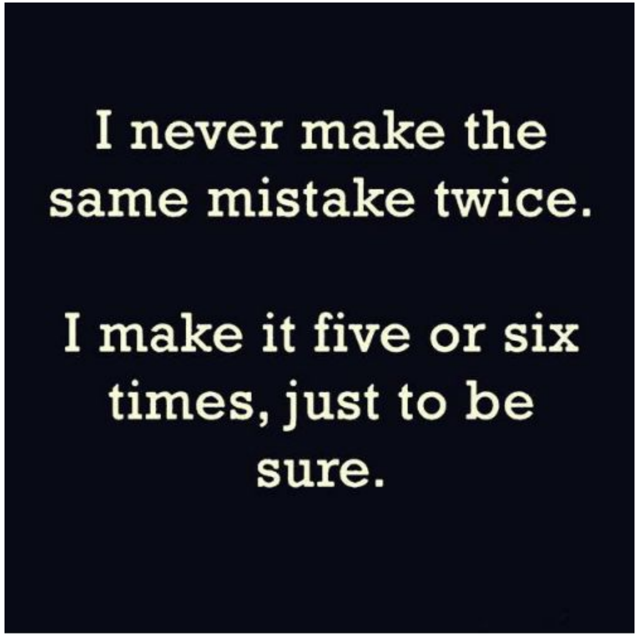
Push Models

- Push moves some credentials out of the trust domain of your cluster into the CI-CD tool



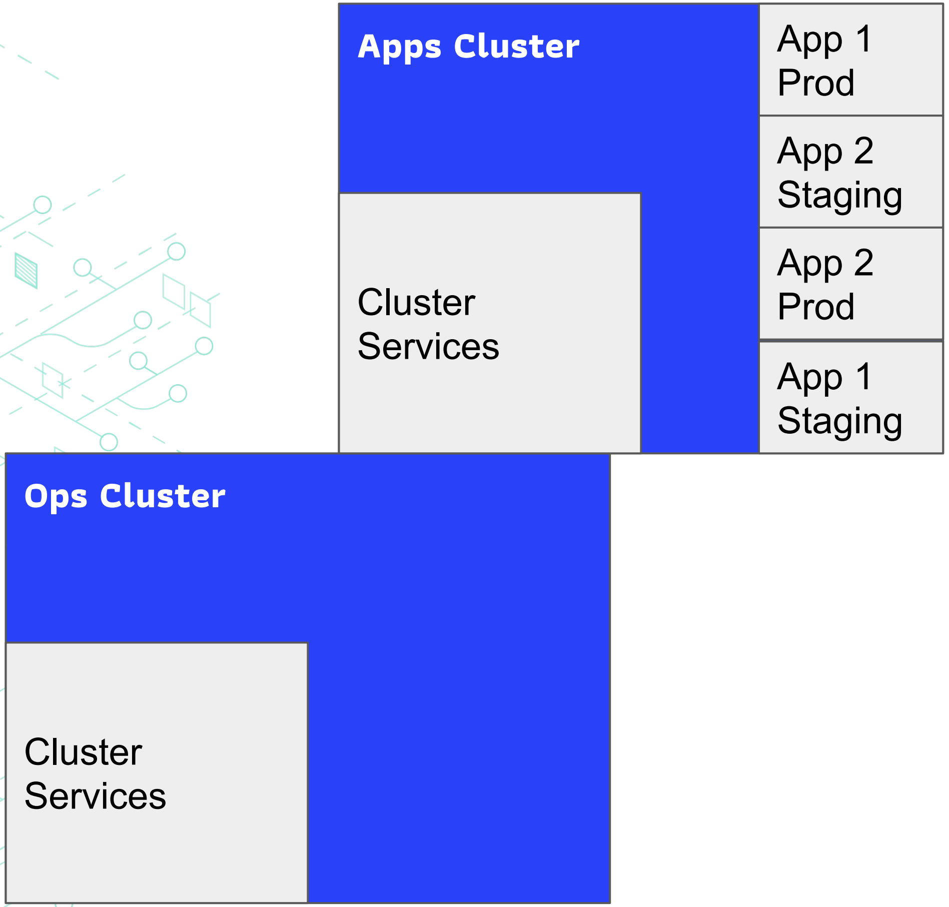
Good Ideas found by experience

- Ops/Apps Pair
- 100% declarative
- Simple pipeline


I never make the same mistake twice.

I make it five or six times, just to be sure.

Ops/Apps Cluster Pair



- Example Of Cluster Services:
  - Prometheus
  - Ingress Controller
  - Istio
  - etc….

Oh No! Not Helm!

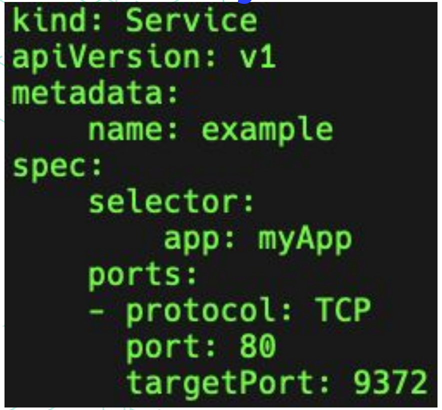
- Helm is not declarative, it is generative

"Look at that beautiful Helm Template!", said No One ever

The Kustomize Approach

- Uses Kubernetes native manifests and allows for an overriding templating approach

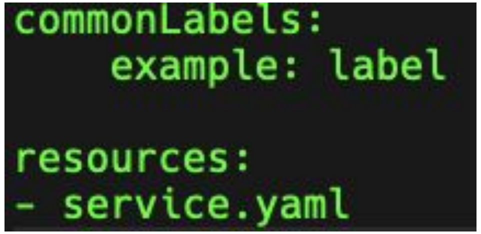
Service.yaml



```
kind: Service
apiVersion: v1
metadata:
    name: example
spec:
    selector:
        app: myApp
    ports:
    - protocol: TCP
      port: 80
      targetPort: 9372
```

+

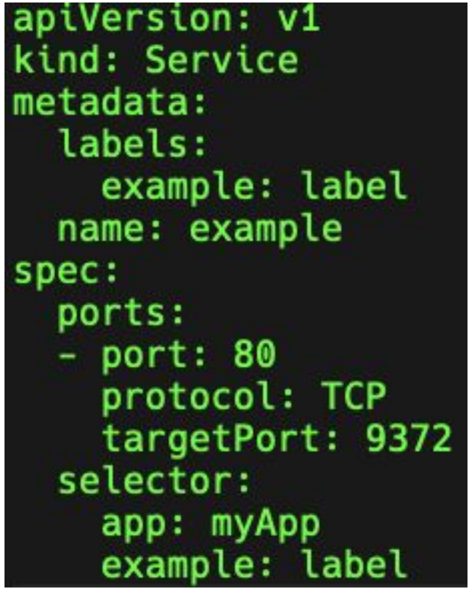
Kustomization.yaml



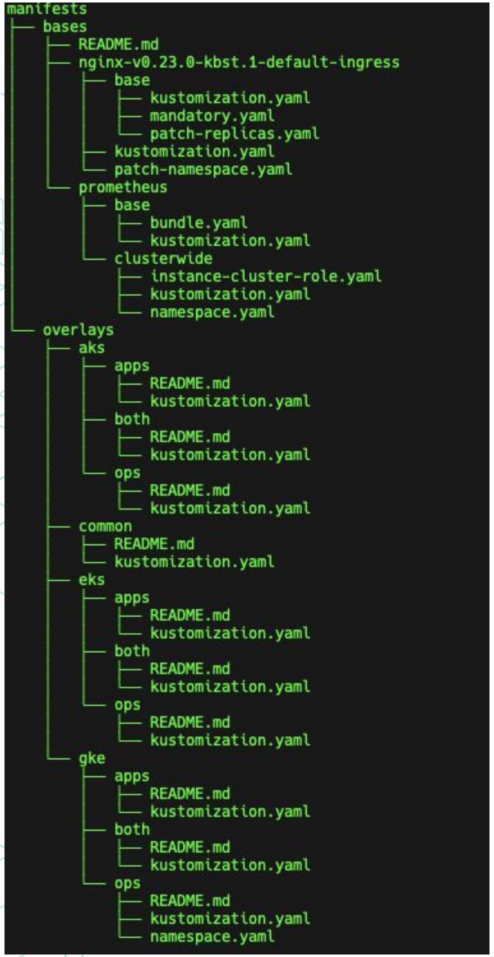
```
commonLabels:
    example: label

resources:
- service.yaml
```

=



```
apiVersion: v1
kind: Service
metadata:
  labels:
    example: label
  name: example
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 9372
  selector:
    app: myApp
    example: label
```



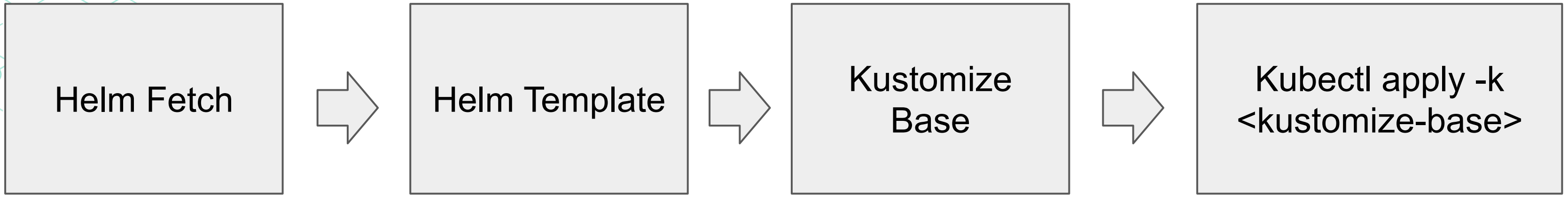
```
manifests
├── bases
│   ├── README.md
│   ├── nginx-v0.23.0-kbst.1-default-ingress
│   │   ├── base
│   │   │   ├── kustomization.yaml
│   │   │   ├── mandatory.yaml
│   │   │   └── patch-replicas.yaml
│   │   ├── kustomization.yaml
│   │   └── patch-namespace.yaml
│   └── prometheus
│       ├── base
│       │   ├── bundle.yaml
│       │   └── kustomization.yaml
│       └── clusterwide
│           ├── instance-cluster-role.yaml
│           ├── kustomization.yaml
│           └── namespace.yaml
└── overlays
    ├── aks
    │   ├── apps
    │   │   ├── README.md
    │   │   └── kustomization.yaml
    │   ├── both
    │   │   ├── README.md
    │   │   └── kustomization.yaml
    │   └── ops
    │       ├── README.md
    │       └── kustomization.yaml
    ├── common
    │   ├── README.md
    │   └── kustomization.yaml
    ├── eks
    │   ├── apps
    │   │   ├── README.md
    │   │   └── kustomization.yaml
    │   ├── both
    │   │   ├── README.md
    │   │   └── kustomization.yaml
    │   └── ops
    │       ├── README.md
    │       └── kustomization.yaml
    └── gke
        ├── apps
        │   ├── README.md
        │   └── kustomization.yaml
        ├── both
        │   ├── README.md
        │   └── kustomization.yaml
        └── ops
            ├── README.md
            ├── kustomization.yaml
            └── namespace.yaml
```

Kustomize

- Use the overlay directories to specify cluster specific setup
- Apps ⇒ Applied to Apps cluster
- Ops ⇒ Applied to Ops cluster
- Both ⇒ Goes to both (duh)

If you Really Must Use Helm

- The Helm + Kustomize Workflow



Tech Stack for this Demo

GitLab

Terraform

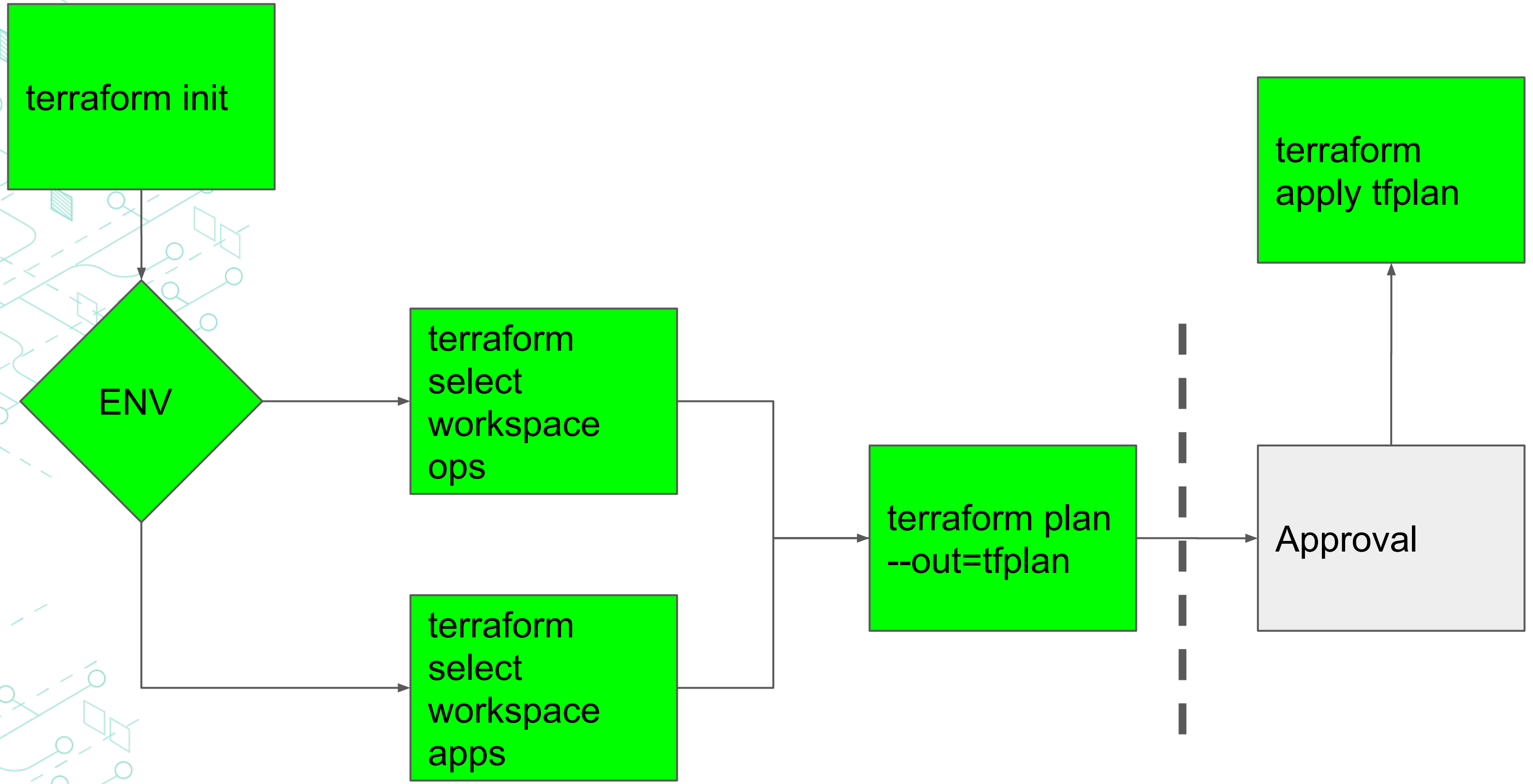
KUBESTACK

Google Kubernetes Engine

Kube Stack

- Methodology on how to structure you infrastructure
- Fits in well with GitOps
- Gives a quick path to production grade Kubernetes setup on managed services (GKE, AKS, EKS, etc...)

General Flow


terraform init
ENV
terraform select workspace ops
terraform select workspace apps
terraform plan --out=tfplan
Approval
terraform apply tfplan

References

- Kube Stack: https://www.kubestack.com/
- Weave Works: https://www.weave.works/blog/gitops-operations-by-pull-request
- Kubediff: https://github.com/weaveworks/kubediff